

15

Tokenization

A token is a representation of an object. We use tokens in many different fields, including economics and computing. In daily life, tokens have been used to represent something of value, such as a gift voucher that is redeemable in exchange for items. In computing, different types of tokens are used, which can be defined as objects that represent eligibility to perform some operation. For example, an access token is used to identify a user and its privileges on a computer system. A hardware security token is used in computer systems to provide a means to authenticate a user (verify their identity) to a computer system. Tokens are also used in computer security mechanisms to substitute sensitive data with non-sensitive equivalents to prevent direct access to sensitive information. For example, in mobile payment systems, tokens are used to safeguard credit card information, which is represented by a token on the mobile device instead of the actual credit card data.

Just like the general use of tokens in day-to-day life, in the blockchain world, tokens also represent something of value. However, the critical difference is that the token exists digitally on a blockchain and is operated cryptographically, which means that they are generated, protected, and transferred using cryptographic protocols. Now that we understand what a token is, we can define tokenization as a process that converts an asset to a digital token on a blockchain. Specifically, it is a process of converting the ownership rights of a real-world asset into a cryptographic/digital token on a blockchain.

In this chapter, we will cover blockchain tokens (or crypto-tokens) and tokenization. We will also cover the following topics:

- Tokenization on a blockchain
- Types of tokens
- Process of tokenization
- Token offerings
- Token standards
- Building an ERC-20 token
- Emerging concepts

Now, we'll begin our discussion on tokenization in the context of blockchain, which is the main aim of this chapter.

Tokenization on a blockchain

Tokenization, in the context of blockchain, is the process of representing an asset digitally on a blockchain. It can be used to represent commodities, real estate, ownership of art, currency, or anything else of value.



Remember, for the rest of the chapter when we refer to tokenization, it is in the context of blockchain.

After this brief definition, let's explore how tokenization can be beneficial.

Advantages of tokenization

Tokenization provides several benefits. The following are some of the most important of these benefits:

- **Faster transaction processing:** As transactions and all relevant parties are present on the blockchain and readily available, there is no need to wait for a response from a counterparty or to wait for clearing and settlement operations. All these operations can be performed efficiently and quickly on a blockchain.
- **Flexibility:** Due to the worldwide adoption of systems that use tokens, such as payment systems, tokenization becomes simple due to easier cross-border use.
- **Low cost:** In comparison with traditional financial products, tokenization requires a lower cost to implement and incurs a lower cost to the end user due to digitization. More recently, the introduction of digital payments has revolutionized the financial industry. In the same spirit, tokenization using blockchain can be considered another step toward achieving further efficiency and cost reduction. In fact, tokenization gives rise to a better, more efficient, and more democratic financial system. For example, just being able to share customer data recorded on one blockchain across all financial institutions reduces costs. Similarly, the possibility of converting illiquid assets into liquid assets is another way of increasing efficiency and profits.
- **Decentralization:** Tokens are presented on a public blockchain, leveraging the decentralization offered by blockchain technology. However, in some cases, a level of somewhat acceptable centralization is introduced due to the control and scrutiny required by investors and exchanges and other involved and interested parties.



We discussed the benefits of decentralization in detail in *Chapter 2, Decentralization*. Refer to that chapter to review decentralization.

- **Security:** As tokens are cryptographically protected and produced using a blockchain, they are secure. However, note that proper implementation must use good practice and meet established security industry standards, otherwise security flaws may result in exploitation by hackers, leading to financial loss.
- **Transparency:** Because they are on a blockchain, tokens are more transparent than traditional financial systems, meaning that any activity can be readily audited on a blockchain and is visible to everyone.
- **Trust:** As a result of the security and transparency guarantees, more trust is naturally placed in tokenization by investors.
- **Fractional ownership:** Imagine a scenario in which you own a painting by Vincent van Gogh. It is almost impossible in traditional scenarios to have multiple owners of the painting without immense legal, financial, and operational challenges. However, on a blockchain using tokens, any asset (such as our van Gogh painting) can be fractionalized in such a way that it can be owned in part by many investors. The same situation is true for a property: if I wanted to have shared ownership with someone, it requires complicated legal procedures. However, with tokenization, fractional ownership of any asset, be it art, real estate, or any other off-chain real-world asset, is quick, easy, efficient, secure, and a lot less complicated than traditional methods.
- **Low entry barrier:** Traditional financial systems require traditional verification mechanisms, which can take a long time for a customer. While they are necessary and serve the purpose in traditional financial systems, these processes take a long time due to the necessary verification processes and the involvement of different entities. However, on a blockchain, this entry barrier is lowered because there is no need to go through the long verification checks. This is because not only is tokenization based on cryptographic guarantees provided by the blockchain, but for many **decentralized applications (DApps)** in this ecosystem, it is basically just a matter of downloading a DApp on your mobile device, depositing some funds if required, and becoming part of the ecosystem.
- **Innovative applications:** Tokenization has resulted in many innovative applications, including novel lending systems on blockchain, insurance, and other numerous financial applications, including decentralized exchanges. Securities can now be tokenized and presented on blockchain, which results in client trust and satisfaction because of the better security and faster processing times.
- **More liquidity:** This is due to the easy availability and accessibility of the tokens for the general public. By using tokens, even illiquid assets such as paintings can be tokenized and traded on an exchange with fractional ownership.

With all these advantages, there are, however, some issues that must be addressed in the tokenization ecosystem. We'll discuss these next.

Disadvantages of tokenization

In this section, we present some of the disadvantages of tokenization. At the top of the list we have regulatory requirements:

- **Regulatory issues:** Regulation is a crucial subject of much debate. It is imperative that the tokens are regulated so that investors can have the same level of confidence that they have when they invest using traditional financial institutions. The big issue with tokenization and generally any public blockchain technology is that they are mostly decentralized and in cases where no single organization is in control, it becomes quite difficult to hold someone responsible if something goes wrong. In a traditional system, we can go to the regulatory authorities or the relevant ombudsman services, but who is held responsible on a blockchain?

Some of this situation has changed with recent security tokenization standards and legislation, which consider tokens as securities. This means that security tokens will then be treated as securities, and the same legal, financial, and regulatory implications will be placed on these tokens that are applicable in the currently established financial industry. Refer to <https://www.sec.gov/answers/about-lawsshtml.html>, where different laws that govern the securities industry are presented. This helps to increase customer confidence levels and trust in the system; however, there are still many challenges that need to be addressed.

A new type of financial crime might be evolving with this tokenization ecosystem where, instead of using well-known and researched traditional financial fraud methods, criminals may choose to launch a technically sophisticated attack. For an average user, this type of attack is difficult to spot and understand as they are entirely on a blockchain and digitized. New forms of front running and market skewing on decentralized finance platforms is increasingly becoming a concern.

- **Legality of tokens:** This is a concern in some jurisdictions where tokens and cryptocurrency are illegal to own and trade.
- **Technology barrier:** Traditional financial systems have been the norm with brick-and-mortar banks. We are used to that system, but things have changed and are expected to change rapidly with tokenization. Tokenization-based financial ecosystems are easier to use for a lot of people, but for some, technological illiteracy can become an issue and could become a barrier. Sometimes the interfaces and software applications required to use tokenization platforms such as trading platforms are difficult to use for the average user.
- **Security issues:** The underlying blockchain technology is considered solid from a security point of view, and it is boasted sometimes that due to the use of cryptography, it is impossible to launch attacks and commit fraud on a blockchain. However, this is not entirely true, even with the firm security foundation that blockchains provide. The way tokenization platforms and DApps are implemented on the blockchain result in security issues that can be exploited by hackers, potentially leading to financial loss. In other words, even if the underlying blockchain is relatively secure, the tokenization DApp implemented on top may have vulnerabilities that could be exploited. These weaknesses could have been introduced by poor development practices or inherent limitations in the still-evolving smart contract languages.

In this section, we have looked at some of the pros and cons of tokenization. Next, let's look at some of the many types of tokens.

Types of tokens

With the rapid development of blockchain technology and the related applications, there has been a tremendous increase in the development of various types of token and relevant ecosystems. First, let's clarify the difference between a coin and a token. Is a Bitcoin a token? Or are tokens and coins the same thing?

A coin is a native token of a blockchain. It is the default cryptocurrency of the blockchain on which it runs. Common examples of such a token are Bitcoin and ether. Both tokens or coins have their own native blockchain on which they run: the Bitcoin blockchain and the Ethereum blockchain.

On the other hand, a token is a representation of an asset that runs on top of a blockchain. For example, Ethereum not only has its own ether cryptocurrency as a native token (or coin) but also has thousands of other tokens that are built on top of Ethereum for different applications. Thanks to its support of smart contracts, Ethereum has become a platform for all sorts of different types of tokens ranging from simple utility tokens, which are usually non-mineable and allow users to perform some specific services on products on the blockchain network/ecosystem, to game tokens (used for gaming) and high-value, application-specific tokens.

Now that we understand the difference between a coin and a token, let's have a look at the usage and significance of different types of tokens. Tokens can be divided broadly into two categories based on their usage: **fungible** tokens and **non-fungible** tokens.

Fungible tokens

From an economics perspective, fungibility is the interchangeability of an asset with other assets of the same type. For example, a ten-pound note is interchangeable with a different ten-pound note or two five-pound notes. It does not matter which specific denominations they are, as long as they are of the same **type** (pound) and have the same **value** (the sum of two five-pound notes), the notes are acceptable.

Fungible tokens work on the same principle. They are:

- **Indistinguishable:** Tokens of the same type are indistinguishable from each other. In other words, they are identical.
- **Interchangeable:** A token is fully interchangeable with another token of the same value.
- **Divisible:** Tokens are divisible into smaller fractions.

Non-fungible tokens

Let's consider **non-fungible tokens** (NFTs), also called *nifty*, with the help of an example. We saw that fungibility allows the same types of token to be interchangeable, but NFTs are not interchangeable. For example, a collectible painting is a non-fungible asset, as it cannot be replaced with another painting of the same type and value. Each painting is unique and distinguishable from others.

NFTs are:

- **Unique:** NFTs are unique and different from other tokens of the same type or in the same category.
- **Non-interchangeable:** Since they are unique and represent specific attributes, these tokens are not interchangeable with tokens of the same type. For example, a rare painting is unique due to its attributes and is not interchangeable with another, even an exact-looking replica. We can also think about the certificate of authenticity that comes with a rare painting; that is also non-interchangeable with another due to its unique attributes representing the rare art.
- **Indivisible:** These tokens are available only as a complete unit. For example, a college diploma is a unique asset that's distinguishable from other diplomas of the same type. It is associated with a unique individual and is thus not interchangeable and it is not rational for it to be divided into fractions, making it indivisible.



One of the prime examples of NFTs is the game CryptoKitties—<https://www.cryptokitties.co>. CryptoKitties played a big role in the popularity of NFTs, in that it was the first game (DApp) that was based on NFTs. From this, the community grew, as did users' interest in the underlying NFT mechanism. A popular term, "crypto collectibles," also emerged with this development. Some other projects that use NFT and offer crypto collectibles include Gods Unchained, Decentraland, Skyweaver, and My Crypto Heroes.

Since the previous edition of this book, NFT popularity has skyrocketed. We will discuss them in more detail in *Chapter 21, Decentralized Finance*.

In the next section, we introduce another interesting topic, stable tokens, which are tokens with interesting properties that make them more appealing to some investors.

Stable tokens

Stable tokens or stable coins are a type of token that has its value pegged to another asset's value, such as fiat currencies or precious metals. Stable tokens maintain a stable price against the price of another asset. Common examples of stable tokens are USDT, developed in 2014 by Tether Limited, and USDC, introduced in 2018 by Coinbase and Circle.

Bitcoin and other similar cryptocurrencies are inherently volatile and experience dramatic fluctuations in their price. This volatility makes them unsuitable for usual everyday use. Stable tokens emerged as a solution to this limitation in tokens.

The price stability is maintained by backing the token up with a stable asset. This is known as collateralization. Fiat currencies are stable because they are collateralized by some reserve, such as **foreign exchange (forex)** reserves or gold. Moreover, sound monetary policy and regulation by authorities play a vital role in the stability of a currency. Tokens or cryptocurrencies lack this type of support and thus suffer from volatility.

There are four types of stable coin:

- **Fiat collateralized:** These stable tokens are backed by or pegged to a traditional fiat currency such as US dollars. Fiat collateralized coins are the most common type of stable coin. Some of the common stable coins available today are **Gemini Dollar (GUSD)** (<https://gemini.com/dollar>), **Paxos (PAX)** (<https://www.paxos.com/pax/>), **USDT (Tether)** (<https://tether.to/en/>), and **Libra** (<https://libra.org/>).
- **Commodity collateralized:** As the name suggests, these stable coins are backed up by fungible commodities (assets) such as oil or gold. Common examples of such tokens are Digix gold tokens (<https://www.crunchbase.com/organization/digix-global>) and Tether gold (<https://tether.to>).
- **Crypto collateralized:** This type of stable coin is backed up by another cryptocurrency. For example, the Dai stable coin (<https://makerdao.com/en/>) accepts Ethereum-based assets as collateral in addition to soft pegging to the US dollar.
- **Algorithmically stable:** This type of stable token is not collateralized by a reserve but stabilized by algorithms that keep track of market supply and demand. For example, Basecoin (<https://basecoin.cc>) can have its value pegged against the US dollar, a basket (a financial term for group, or collection of similar securities) of assets, or an index, and it also depends on a protocol to control the supply of tokens based on the exchange rate between itself and the peg used, such as the US dollar. This mechanism helps to maintain the stability of the token.

Security tokens

Security tokens derive their value from external tradeable assets. For example, security tokens can represent shares in a company. The difference is that traditional security is kept in a bank register and traded on traditional secondary markets, whereas security tokens are available on a blockchain. Being securities, they are governed by all traditional laws and regulations that apply to securities, but due to their somewhat decentralized nature, in which no middleman is required, security tokens are seen as a more efficient, scalable, and transparent option.

Now that we have covered different types of token, let's see how an asset can be tokenized by exploring the process of tokenization.

Process of tokenization

In this section, we'll present the process of tokenization, discuss what can be tokenized, and provide a basic example of the tokenization of assets.

Almost any asset can be tokenized and presented on a blockchain, such as commodities, bonds, stocks, real estate, precious metals, loans, and even intellectual property. Physical goods that are traditionally illiquid in nature, such as collectibles, intellectual property, and art, can also be tokenized and turned into liquid tradeable assets.

A generic process to tokenize an asset or, in other words, offer a security token, is described here:

1. The first step is to onboard an investor who is interested in tokenizing their asset.

2. The asset that is presented for tokenization is scrutinized and audited, and ownership is confirmed. This audit is usually performed by the organization offering the tokenized security. It could be an exchange or a cryptocurrency start-up.
3. The process of tokenized security starts, which leads to the **security token offering (STO)**.
4. The physical asset is placed with a custodian (in the real world) for safekeeping.
5. The **derivative token**, representing the asset, is created by the organization offering the token and issued to investors through a blockchain.
6. Traders start to buy and sell these tokens using trading exchanges in a secondary market and these trades are settled (the buyer makes a payment and receives the goods) on a blockchain.

Common platforms for tokenization include Ethereum, Solana, and EOS. Tezos is also emerging as another platform for tokenization due to its support for smart contracts. In fact, any blockchain that supports smart contracts can be used to build tokens; however, the most common are Ethereum and Solana.

At this point, a question arises about how all these different types of tokens reach the public for investment. In the next section, we examine how this is achieved by first explaining what token offerings are and examining the different types.

Token offerings

Token offerings are mechanisms to raise funds and profit. There are a few different types of token offerings. We will introduce each of these separately now. One main common attribute of each of these mechanisms is that they are hosted on a blockchain and make use of tokens to facilitate different financial activities. These financial activities can include crowdfunding and trading securities.

Initial coin offerings

Initial coin offering or **initial currency offering (ICO)** is a mechanism for raising funds using cryptocurrencies. ICOs have been a very successful but somewhat controversial mechanism for raising capital for new cryptocurrency or token projects. ICOs are somewhat controversial sometimes due to bad design or poor governance, but at times some of the ICOs have turned out to be outright scams. A list of fraudulent schemes is available at <https://dfpi.ca.gov/crypto-scams/>. These incidents have contributed to the bad reputation and controversy surrounding ICOs in general. While there are some scams, there are also many legitimate and successful ICOs. The **return on investment (ROI)** for quite a few of ICOs has been quite big and has contributed toward the unprecedented success of many of these projects. Some of these projects are Ethereum, NXT, NEO, IOTA, and QTUM.

A common question is asked here regarding the difference between traditional **initial public offerings (IPOs)** and **ICOs**, as both mechanisms are fundamentally designed to raise capital. The difference is simple: ICOs are blockchain-based token offerings that are usually initiated by start-ups to raise capital by allowing investors to invest in their start-up. Usually in this case, contributions by investors are made using already existing and established cryptocurrencies such as Bitcoin or ether. As a return, when the project comes to fruition, the initial investors get their return on the investment.

On the other hand, IPOs are traditional mechanisms used by companies to distribute shares to the public. This is done using the services of underwriters, which are usually investment banks. IPOs are only usually allowed for those companies that are already well established, but ICOs on the other hand can be offered by start-ups. IPOs also offer dividends as returns, whereas ICOs offer tokens that are expected to rise in value once the project goes live.

Another key comparison is that IPOs are regulated, traditional mechanisms that are centralized in nature, while ICOs are decentralized and unregulated.



There are some examples where traditional means such as signing paper contracts have been used to provide some level of legal protection for the parties involved, but ICOs are largely unregulated. This is especially true from the perspective of large financial regulatory bodies and ombudsman services providing investor protection and handling consumer complaints in the traditional financial services industry.

Because ICOs have been unregulated, which has resulted in a number of scams and people losing their money, another form of fundraising known as security token offerings was introduced. We'll discuss this next.

Security token offerings

Security token offerings (STOs) are a type of offering in which tokenized securities are traded at cryptocurrency exchanges. Tokenized securities or security tokens can represent any financial asset, including commodities and securities. The tokens offered under STOs are classified as securities. As such, they are more secure and can be regulated, in contrast with ICOs. If an STO is offered on a traditional stock exchange, then it is known as a tokenized IPO. Tokenized IPO is another name for an STO that is used when an STO is offered on a regulated stock exchange, which helps to differentiate between an STO offered on a traditional regulated exchange and one that is offered on cryptocurrency exchanges. STOs are regulated under the Markets in Financial Instruments Directive—MiFID II—in the European Union.

Initial exchange offerings

Initial exchange offering (IEO) is another innovation in the tokenization space. The key difference between an IEO and an ICO is that in an ICO, the tokens are distributed via crowdfunding mechanisms to investors' wallets, but in an IEO, the tokens are made available through an exchange.

IEOs are more transparent and credible than ICOs due to the involvement of an exchange and due diligence performed by the exchange.

Equity token offerings

Equity token offerings (ETOs) are another variation of ICOs and STOs. ICOs offer utility tokens, whereas equity tokens are offered in ETOs. When compared with STOs, ETOs offer shares of a company, whereas STOs offer shares in any asset, such as currencies or commodities. From this point of view, ETOs can be regarded as a specific type of STO, where only shares in a company or venture are represented as tokens.

Decentralized autonomous initial coin offering

Decentralized Autonomous Initial Coin Offering (DAICO) is a combination of **decentralized autonomous organizations (DAOs)** and ICOs that enables investors to have more control over their investment and is seen as a more secure, decentralized, and automated version of ICOs.

Other token offerings

Different variations of ICOs and new concepts are being introduced quite regularly, and innovation is only expected to grow in this area.

A comparison of different types of token offering is presented in the following table:

Name	Concept	First introduced	Scale of decentralization	How to invest	Regulation
ICO	Crowdfunding through a utility token	In July 2013 with Mastercoin	Semi-decentralized	Investors send cryptocurrency to a smart contract released by the token offerer	Low regulation
STO	Tokenized security such as bonds and stocks	2017	Somewhat centralized	Use the exchange-provided platform	Regulated under established laws and directives in many jurisdictions
IEO	Tokens are made available through an exchange	2018	Somewhat centralized	Use the exchange-provided platform	Low regulation
ETO	Offers shares of any asset	December 2018 with the Neufund ETO	Somewhat centralized	Use the exchange-provided platform	Mostly regulated
DAICO	Combination of DAO and ICO	May 2018 with ABYSS DAICO	Mostly decentralized	Investors send cryptocurrency to the DAICO smart contract	Low regulation

With all these different types of tokens and associated processes, a need to standardize naturally arises so that more adoption and interoperability can be achieved. To this end, several development standards have been proposed, which we discuss next.

Token standards

With the advent of smart contract platforms such as Ethereum, it has become quite easy to create a token using a smart contract. Technically, a token or digital currency can be created on Ethereum with a few lines of code, as shown in the following example:

```
pragma solidity ^0.8.0;
contract token {
    mapping (address => uint) public coinBalanceOf;
    event CoinTransfer(address sender, address receiver, uint amount);

    /* Initializes contract with initial supply tokens to the creator of the
    contract */
    function tokenx(uint supply) public {
        supply = 1000;
        coinBalanceOf[msg.sender] = supply;
    }
    /* Very simple trade function */
    function sendCoin(address receiver, uint amount) public returns(bool
sufficient) {
        if (coinBalanceOf[msg.sender] < amount) return false;
        coinBalanceOf[msg.sender] -= amount;
        coinBalanceOf[receiver] += amount;
        emit CoinTransfer(msg.sender, receiver, amount);
        return true;
    }
}
```



This code is based on one of the early codes published by Ethereum as an example on ethereum.org.

The preceding code works and can be used to create and issue tokens. However, the issue is that without any standard mechanism, everyone would implement tokenization smart contracts in their own way based on their requirements. This lack of standardization will result in interoperability and usability issues, which obstruct the widespread adoption of tokenization.

To remedy this, the first tokenization standard was proposed on Ethereum, known as ERC-20.

ERC-20

ERC-20 is the most famous token standard on the Ethereum platform. Many token offerings are based on ERC-20 and there are wallets available, such as MetaMask, that support ERC-20 tokens.

ERC-20 was introduced in November 2015 and since then has become a very popular standard for fungible tokens. This standard has been used in many ICOs and has resulted in valuable digital currencies (tokens) over the last few years, including EOS, Golem, and many others. Famous tokens, such as USDT, BNB, USDC, and Matic, are all based on the ERC-20 standard. There are almost 1,000 ERC-20 token projects listed on Etherscan (<https://etherscan.io/tokens>), which is a clear indication of this standard's popularity.

While ERC-20 defined a standard for fungible tokens and was widely adopted, it has some flaws, which result in some security and usability issues. For example, a security issue in ERC-20 results in a loss of funds if the tokens are sent to a smart contract that does not have the functionality to handle tokens. The effectively “burned” tokens result in a loss of funds for the user.

To address these shortcomings, ERC-223 was proposed.

ERC-223

ERC-223 is a fungible token standard. One major advantage of ERC-223 as compared to ERC-20 is that it consumes only 50% of ERC-20's gas consumption, which makes it less expensive to use on Ethereum's main net. ERC-223 is backward compatible with ERC-20 and is used in a number of major token projects such as LINK and CNexchange (CNEX).

ERC-777

The main aim of ERC-777 is to address some of the limitations of ERC-20 and ERC-223. It is backward compatible with ERC-20. It defines several advanced features to interact with ERC-20 tokens. It allows sending tokens on behalf of another address (contract or account). Moreover, it introduces a feature of “hooks,” which allows token holders to have more control over their tokens.



The Ethereum Improvement Proposal (EIP) is available here: <https://eips.ethereum.org/EIPS/eip-777>

ERC-721

ERC-721 is an NFT standard. ERC-721 mandates several rules that must be implemented in a smart contract for it to be ERC-721 compliant. These rules govern how these tokens can be managed and traded. ERC-721 was made famous by the **CryptoKitties** project. CryptoKitties is a blockchain game that allows players to create (breed) and trade different types of virtual cats on the blockchain. Each “kitty” is unique and tradeable for a value on the blockchain.

ERC-884

This is the standard for ERC-20-compliant share tokens that are conformant with Delaware General Corporations Law.



The legislation is available here: <https://legis.delaware.gov/json/BillDetail/GenerateHtmlDocument?legislationId=25730&legislationTypeId=1&docTypeId=2&legislationName=SB69>

The token standard EIP is available here: <https://github.com/ethereum/EIPs/blob/master/EIPS/eip-884.md>

ERC-1400

This is a security token standard that defines how to build, issue, trade, and manage security tokens. Under ERC-1400, there are a few other standards, which are as follows:

- **ERC-1410:** A partially fungible token standard that defines a standard interface for grouping an owner's tokens into partitions
- **ERC-1594:** This defines a core security token standard
- **ERC-1643:** This is the document management standard
- **ERC-1644:** This is the token controller operations standard
- **ERC-1066:** This standard specifies a standard way to design Ethereum status codes

The aim of ERC-1400 and the standards within it is to cover all activities related to the issuance, management, control, and processing of security tokens.

ERC-1404

This ERC standard allows the issuance of tokens with regulatory transfer restrictions. These restrictions enable users to control the transfer of tokens in different ways. For example, users can control when, to whom, and under what conditions the tokens can be transferred. For example, an issuer can choose to issue tokens only to a whitelisted recipient or check whether there are any timing restrictions on the senders' tokens.

ERC-1404 introduces two new functions in the ERC-20 standard to introduce restriction and control mechanisms. These two functions are listed as follows:

```
contract ERC1404 is ERC20 {  
    function detectTransferRestriction (address from, address to, uint256 value)  
    public view returns (uint8);  
    function messageForTransferRestriction (uint8 restrictionCode) public view  
    returns (string);  
}
```



More information on ERC-1404 is available at <https://erc1404.org>

ERC-1155

This is the multi-token standard. This standard allows bundling of transactions to save costs and more efficient trades. It can be used to create utility tokens and NFTs. The idea is to create a smart contract interface that can handle and represent any number of fungible and NFT types. This means that ERC-1155 can function as ERC-20 and ERC-721, even at the same time. It also improves the efficiency and correctness of ERC-20 and ERC-721.



More information is available in EIP-1155-Multi token standard at <https://eips.ethereum.org/EIPS/eip-1155>

ERC-4626

This is a tokenized vault standard that aims to optimize and unify the technical parameters of tokenized yield-bearing vaults.



Tokenized yield-bearing vaults are smart contracts that accumulate yield as the token holders lock tokens inside the vault. In other words, they help to maximize the yield tokens holders can earn from their deposited tokens.

Tokenized vaults are used on Ethereum in different protocols, such as Yearn Finance, Tulip Garden, and Pickle Finance. These vaults allow a token holder to deposit their tokens into the vault and earn a yield on those tokens. When a token holder deposits their tokens into a vault, they get a vault token. This vault token can appreciate over time. The token holders can get their original tokens back when they return these vault tokens.

The problem is that there is no single unifying standard for implementing standard operations such as depositing and withdrawing. This difference in implementations creates friction between different protocols and results in integration attempts, which may or may not correctly work. For example, if you want to write a DApp that interacts with all these different tokenized vaults, then you have to write adapters that can talk to all these different vaults, making it difficult for integrators and aggregators. ERC-4626 aims to solve this problem by providing a standard way to interact with these vaults. It provides an interface that standardizes deposits, withdrawals, reading balance operations, and other parameters. This standard will result in interoperability, composability, and improvement of the overall decentralized finance and blockchain ecosystem.



More details on the standard are available here: <https://eips.ethereum.org/EIPS/eip-4626>

With this, we have completed our introduction to ERC standards. Now after all this theoretical background, let's see how a token can be built on Ethereum. We will build our own ERC-20-compliant token.

Building an ERC-20 token

In this section, we will build an ERC-20 token. In previous chapters, we saw several ways of developing smart contracts, including writing smart contracts in Visual Studio Code, and compiling them and then deploying them on a blockchain network. We also used the Remix IDE, Truffle, and MetaMask to experiment with various ways of developing and deploying smart contracts. In this section, we will use a quick method to develop, test, and deploy our smart contract on the Sepolia test network. We will not use Truffle or Visual Studio Code in this example as we have seen this method before; we are going to explore a different and quicker method to build and deploy our contract.

In this example, we will see how quickly and easily we can build and deploy our own token on the Ethereum blockchain network. The aim of this exercise is to understand how MetaMask can be used to deploy our new token smart contract on an Ethereum network. We will also see how we can import ERC-20 tokens in MetaMask and use it to transfer funds from one account to another.

We will use the following components in our example, which were both introduced in *Chapter 10, Ethereum in Practice*:

- **Remix IDE**, available at <http://remix.ethereum.org>. Note that in the following example, the user interface and steps required might be slightly different depending on the stage of development. This is because Remix IDE is under heavy development, and new features are being added to it at a tremendous pace—as such, some changes are expected in the user interface too. However, the fundamentals are not expected to change, and no major changes are expected in the user interface.
- **MetaMask**. We will use the network we set up in *Chapter 10, Ethereum in Practice*, but you can create a new account if required.

Now let's start writing our code.

Building the Solidity contract

First, we explore what the ERC-20 interface looks like, and then we will start writing our smart contract step by step in the Remix IDE.

The ERC-20 interface defines a number of functions and events that must be present in an ERC-20-compliant smart contract. Some more rules that must be present in an ERC-20-compliant token are listed here:

- **totalSupply**: This function returns the number of the total supply of tokens:

```
function totalSupply() public view returns (uint256)
```

- **balanceOf**: This function returns the balance of the token owner:

```
function balanceOf(address _owner) public view returns (uint56 balance)
```

- **transfer:** This function takes the address of the recipient and a specified value as a number of tokens and transfers the amount to the address specified:

```
function transfer(address _to, uint256 _value) public returns (bool success)
```

- **transferFrom:** This function takes `_from` (sender's address), `_to` (recipient's address), and `_value` (amount) as parameters and returns true or false. It is used to transfer funds from one account to another:

```
function transferFrom(address _from, address _to, uint256 _value) public returns (bool success)
```

- **approve:** This function takes `_spender` (recipient) and `_value` (number of tokens) as parameters and returns a Boolean, true or false, as a result. It is used to authorize the spender's address to make transfers on behalf of the token owner up to the approved amount (`_value`):

```
function approve(address _spender, uint256 _value) public returns (bool success)
```

- **allowance:** This function takes the address of the token owner and the spender's address and returns the remaining number of tokens that the spender has approval to withdraw from the token owner:

```
function allowance(address _owner, address _spender) public view returns (uint256 remaining)
```

There are three optional functions, which are listed as follows:

- **name:** This function returns the name of the token as a string. It is defined in code as follows:

```
function name() public view returns (string)
```

- **symbol:** This returns the symbol of the token as a string. It is defined as follows:

```
function symbol() public view returns (string)
```

- **decimals:** This function returns the number of decimals that the token uses as an integer. It is defined as follows:

```
function decimals() public view returns (uint8)
```

Finally, there are two events that must be present in an ERC-20-compliant token:

- **Transfer:** This event must trigger when tokens are transferred, including any zero-value transfers. The event is defined as follows:

```
event Transfer(address indexed _from, address indexed _to, uint256 _value)
```


- **Approval:** This event must trigger when a successful call is made to the approve function. The event is defined as follows:

```
event Approval(address indexed _owner, address indexed _spender, uint256
_value)
```

Now let's have a look at the source code of our ERC-20 token. This is written in Solidity, which we are familiar with and explored in detail in *Chapter 11, Tools, Languages, and Frameworks for Ethereum Developers*.

The source code for our ERC-20 token is as follows—we will explain it step by step before writing it into the Remix IDE.

First is the SPDX license identifier:

```
// SPDX-License-Identifier: GPL-3.0
```

Second, we have the Solidity compiler and language version, which specifies the compiler version using the pragma directive for which our program is written:

```
pragma solidity ^0.8.0;
```

Then we create the contract object:

```
contract MyERC20Token {
```

After this, the contract object is defined with the name MyERC20Token.

Then we have the mappings:

```
mapping (address => uint256) _balances;
mapping (address => mapping(address => uint256)) _allowed;
```

These are the two mappings used in the ERC-20 smart contract. The first one is for keeping balances and the other one is used for allowances.

These are the state variables:

```
string public name = "My ERC20 Token";
string public symbol = "MET";
uint8 public decimals = 0;
uint256 private _totalSupply = 100;
```

They describe the name, symbol, and decimal precision points and the total supply of our token.

This is the Transfer event:

```
event Transfer(address indexed _from, address indexed _to, uint256 _value);
```

It has three parameters: from, to, and value. from represents the address from which the tokens are coming, to is the account to which tokens are being transferred, and value is the number of tokens.

This is the Approval event:

```
event Approval(address indexed _owner, address indexed _spender, uint256 _value);
```

This has three parameters: owner address, recipient address, and value. The indexed keyword allows us to search for a specific log item instead of searching through all logs. It enables log filtration to search and extract only the required data instead of returning all logs.

This is the constructor that is executed when the contract is created:

```
constructor(){  
    _balances[msg.sender] = _totalSupply;  
    emit Transfer(address(0), msg.sender, _totalSupply);  
}
```

It is optional in Solidity and is used to run initialization code. In our example, the initialization code contains the statements to transfer the entire balance, `_totalSupply`, to the creator of the smart contract; in our case, it is the sender account. It also then emits the `Transfer` event, indicating that the transfer has taken place from `address(0)` to `msg.sender` (our contract creator) and `_totalSupply`, which is 100 in our case, just to keep things simple.

This is the `totalSupply` function:

```
function totalSupply() public view returns (uint) {  
    return _totalSupply - _balances[address(0)];  
}
```

This function returns the total amount of tokens after deducting it from the balance of the account.

This is the `balanceOf` function:

```
function balanceOf(address _owner) public view returns (uint balance) {  
    return _balances[_owner];  
}
```

The `balanceOf` function returns the balance of the token owner.

The allowance function is as follows:

```
function allowance(address _owner, address _spender) public view returns (uint remaining) {  
    return _allowed[_owner][_spender];  
}
```

The allowance function returns the total remainder of the tokens.

This is the transfer function, which returns true or false depending on the result of the execution:

```
function transfer(address _to, uint256 _value) public returns (bool success) {
    require(_balances[msg.sender] >= _value, "value exceeds senders balance");
    _balances[msg.sender] -= _value;
    _balances[_to] += _value;
    emit Transfer(msg.sender, _to, _value);
    return true;
}
```

The require convenience function is used to check for certain conditions and throw an exception if the conditions are not met. In our example, require checks whether the value exceeds the sender's balance, and if it does, an error message will be generated stating that value exceeds senders balance. If this check passes, the transfer occurs, and after emitting the Transfer event, the function returns true, indicating the successful transfer of tokens.

This is the approve function, which returns true or false depending on the result of the execution of the function:

```
function approve(address _spender, uint256 _value) public returns (bool
success)
{
    _allowed[msg.sender][_spender] = _value;
    emit Approval(msg.sender, _spender, _value);
    return true;
}
```

This function takes _spender (the user) and _value (number of tokens) as arguments and serves as a mechanism to provide approval to the user to acquire the allowed number of tokens from our ERC-20 contract.

This function is the transferFrom function, which can be used to automate the transfer of tokens from one address to another:

```
function transferfrom(address _from, address _to, uint256 _value) public
returns (bool success)
{
    require(_value <= _balances[_from], "Not enough balance");
    require(_value <= _allowed[_from][msg.sender], "Not enough allowance");
    _balances[_from] -= _value;
    _balances[_to] += _value;
    _allowed[_from][msg.sender] -= _value;
    emit Transfer(_from, _to, _value);
    return true;
}
```

It takes three parameters: `_from`, `_to`, and `_value`. It returns `true` or `false` depending upon the execution of the function. First, with the `require` functions, the balance, and allowances are checked to ensure that enough balance and allowance are available. After that, the transfer occurs, and eventually the `Transfer` event is emitted, followed by a `true` Boolean value returned by the function indicating the successful transfer.

Now that we understand what our source code does, the next step is to write it in the Remix IDE and deploy it.

Deploying the contract on the Remix JavaScript virtual machine

In this step, we simply take the source code and write or simply paste it in the Remix IDE. To achieve this, take the following steps:

1. Open the Remix IDE.
2. When the Remix IDE starts up, select **SOLIDITY** under **Environments**, as we are going to write smart contracts using the Solidity smart contract language.
3. After selecting the Solidity environment, create a new file by choosing the **FILE EXPLORERS** option from the list of icons on the left-hand side and add a new file by clicking the **+** sign.
4. Create a new file in the Remix IDE named `erc20example.sol`.
5. When we have created the new file, simply write the source code in the IDE, or paste it directly.
6. Compile the source code. To do this, click on **Compile erc20example.sol**. Optionally, **Auto compile** can also be selected under **COMPILER CONFIGURATION**, which will compile the code automatically as soon as it is written into the IDE:

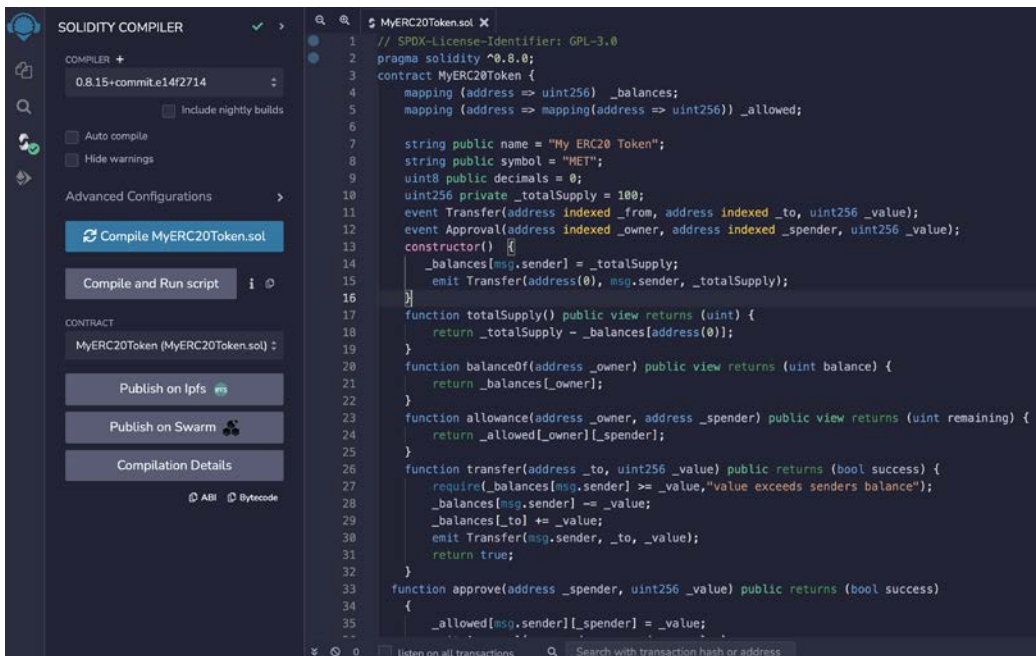


Figure 15.1: Solidity compiler in Remix

7. Once compiled successfully, it is ready to be deployed. First, we will deploy it on JavaScript VM available with the Remix IDE to ensure that everything works. Once we have tested that it can be deployed correctly and works as per our expectations, we could deploy it on main net using MetaMask. In our example, however, we will deploy it on the Sepolia (<https://sepolia.dev>) test net using MetaMask.



To deploy on main net, simply choose main net from MetaMask after ensuring that enough funds are available to deploy it on main net.

Remember that we have some ether left from our exercise in *Chapter 13, The Merge and Beyond*, and have a test account on the Sepolia network. We can use the same account for this example or create a new account and fund it using the process described in the aforementioned chapter. As an alternative, you can use faucets available for the Sepolia network at <https://sepolia-faucet.pk910.de>.

8. After compilation, we deploy the smart contract using the **Deploy & Run Transactions** interface available within the Remix IDE. Make sure the environment JavaScript VM is selected and an account is selected from which to deploy, and then select **Deploy**:

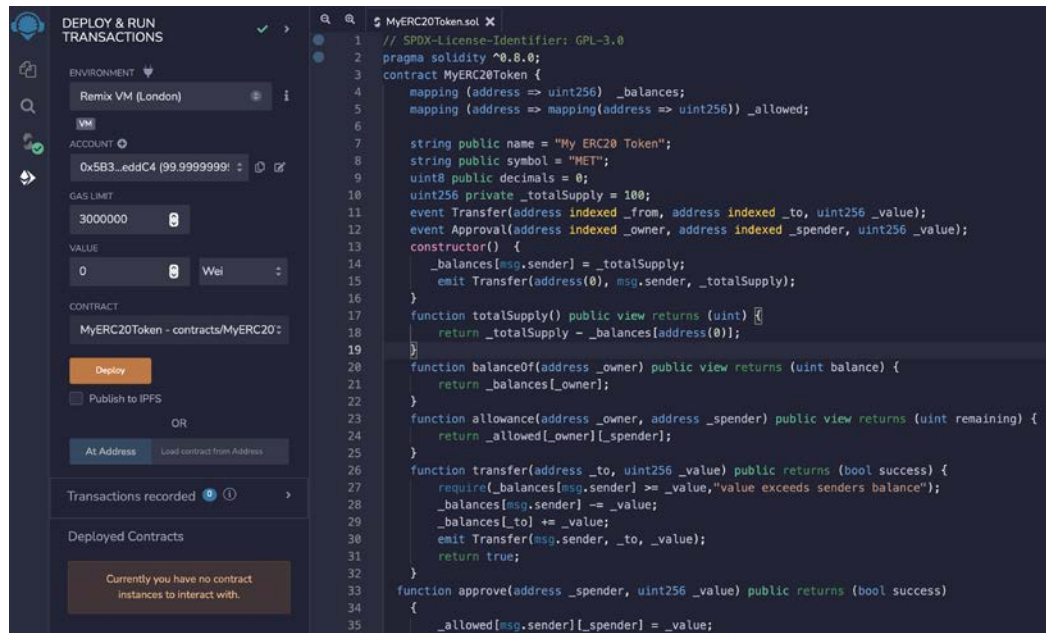


Figure 15.2: Deploying and running transactions in Remix

Once it's deployed, the contract will become available under **Deployed Contracts** and, in the logs, we can see relevant details regarding the deployment of the contract.

Most importantly, we can see in the logs that when the contract is created, the first event emitted is 'Transfer'. Here, we can see that all 100 tokens have been transferred to the owner account, 0xd8b934580fcE35a11B58C6D73aDeE468a2833fa8:

```
logs
[
  {
    "from": "0xd8b934580fcE35a11B58C6D73aDeE468a2833fa8",
    "topic":
"0xddf252ad1be2c89b69c2b068fc378daa952ba7f163c4a11628f55a4df523b3ef",
    "event": "Transfer",
    "args": {
      "0": "0x0000000000000000000000000000000000000000000000000000000",
      "1": "0x5B38Da6a701c568545dCfcB03FcB875f56beddC4",
      "2": "100",
      "_from": "0x00000000000000000000000000000000000000000000000000000",
      "_to": "0x5B38Da6a701c568545dCfcB03FcB875f56beddC4",
      "_value": "100"
    }
  }
]
```

Once it's deployed, we will see under **Deployed Contracts** all the functions exposed by the contract in the Remix IDE:

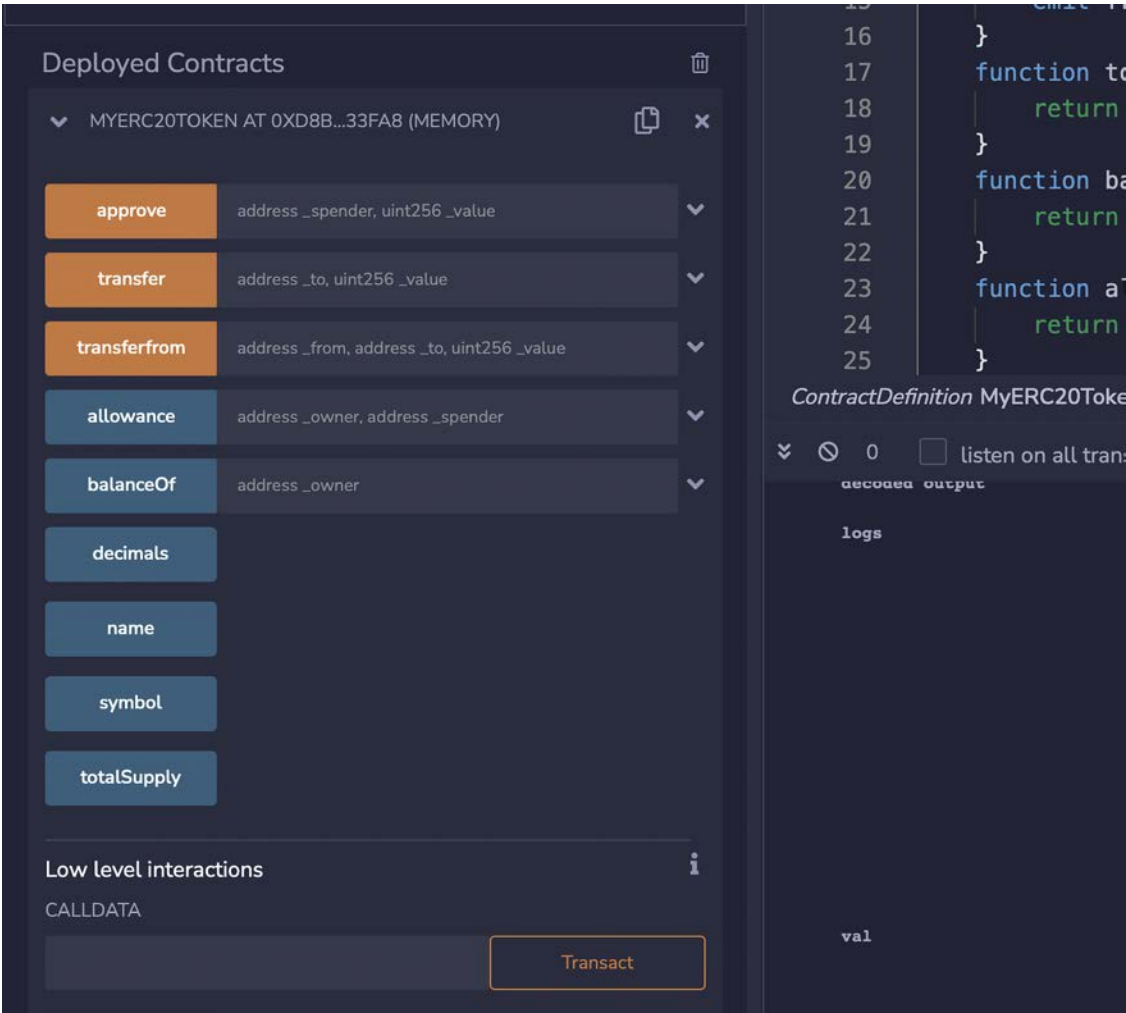


Figure 15.3: Deployed Contracts—exposed functions in Remix for our contract

We can test the functionality of our contract using this interface. For example, calling `totalSupply` shows a value of `100`, which indicates the number of tokens, and calling `symbol` shows the string `MET`, our token’s symbol.

At this point, our contract works, and we have tested it locally. Now we can deploy it onto the Sepolia test network. We will use MetaMask for this purpose:

1. In the Remix IDE, under **DEPLOY & RUN TRANSACTIONS**, select the **Injected Web3** option from the ENVIRONMENTS list. This is the execution environment that is provided by enabling web3 within the browser using the MetaMask plugin.
2. Next, confirm that MetaMask is running and connected to the Sepolia test network. This should be easy to check, as we used this same network in *Chapter 13, The Merge and Beyond*.

If everything is working in MetaMask, it should display a screen similar to the following screenshot. Note that you may have to log in again to MetaMask. Once you're logged in, select the Sepolia network and an account that has some ether in it.

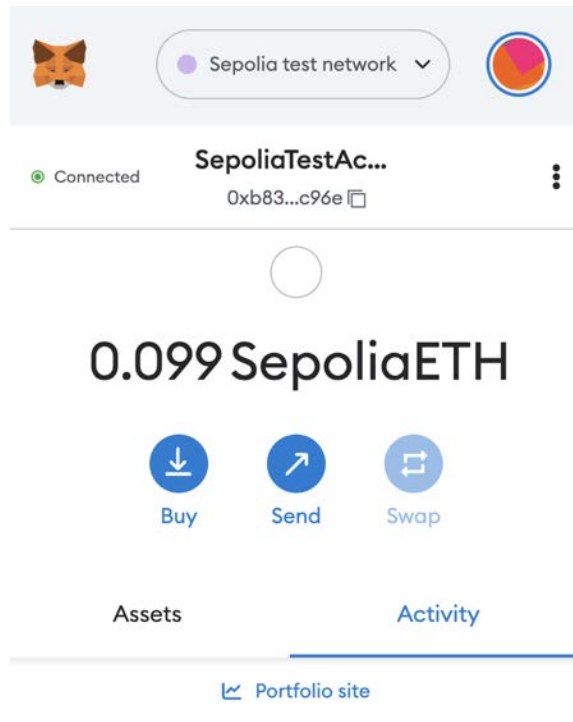


Figure 15.4: Sepolia test network in MetaMask

3. In the Remix IDE, choose **Injected Provider - MetaMask** in the ENVIRONMENT field under **DEPLOY & RUN TRANSACTIONS**. Once connected you will notice that the **Sepolia network** (Sepolia (11155111) network) is shown in remix IDE just under the **Environment** box.

Note that MetaMask also shows that it's connected to the Sepolia network, and it also shows the account that we have set up in MetaMask. Now we are all set to deploy this on the Sepolia test network.

4. Click **Deploy**, which will open the MetaMask window to confirm the transaction.
5. Click **Confirm**, and the contract will be deployed. We can see this in the history in MetaMask. Also, in the Remix IDE logs, we can see if it has been deployed successfully.

Like the tests that we did earlier in this example when we deployed our contract on the JavaScript VM, we can invoke different functions exposed by our new ERC-20 token directly from the Remix IDE. We can also see our token on the Etherscan token tracker. This is shown via <https://sepolia.etherscan.io/tx/0xfd427145a393fb6dbb08bf9f3e4c945acb1039404503380692b81b366b0f23e6>.

The Etherscan page also shows detailed information about our token contract such as total supply, contract address, etc, which can be accessed here : <https://sepolia.etherscan.io/token/0x07c152a6ab577e8f78e3bede502d79652787a9fc>.

So here we have it, our own MET token deployed on the Sepolia test network. Now, if desired, we could deploy this to the Ethereum main net if we have some ether available. We can perform the same steps to deploy it, with the only difference being that in MetaMask, we will choose the Ethereum main net as the network.

Adding tokens in MetaMask

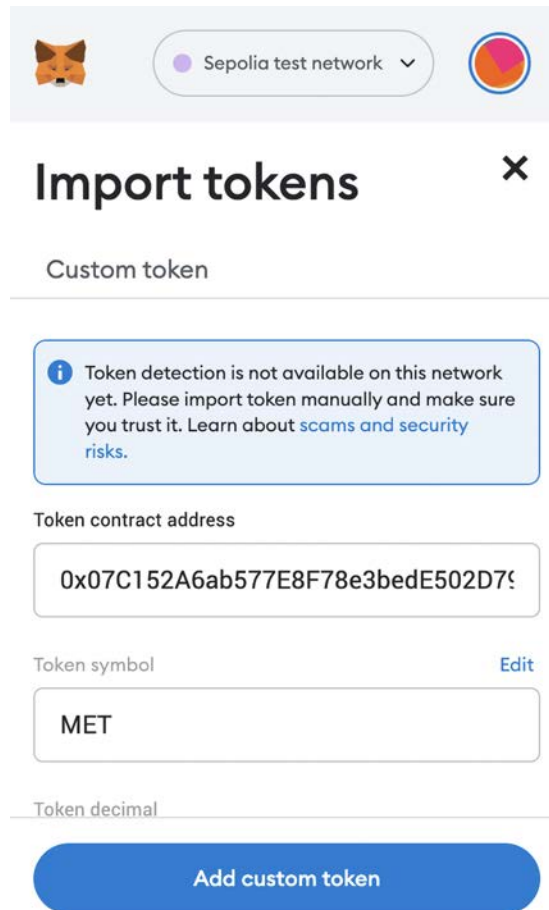
Once we have deployed our contract and our ERC-20 token is now on the blockchain (so to speak), unless we are able to view it and perform operations on it, the token on its own is of no real use.

To perform operations on the token, we can manually create commands using the Web3.js `sendRawTransaction` method and use the JavaScript and command-line `geth` console to interact with it. Alternatively, we can use an easier option and simply add the token to a wallet. Wallets abstract away the complexities associated with transaction creation and management and provide an easy-to-use interface to perform transfers and similar tasks.

MetaMask can serve as a wallet for tokens and provides an interface to add tokens. In this section, we'll see how we can add our MET token to MetaMask and perform some operations on it:

1. Open MetaMask and find the **Add Token** option on the main user interface where it says *Don't see your token? Import tokens* and click on **import token**. Alternatively, open the account menu and it will display the account import option.
2. Click on **Add Token** and select **Custom Token**.

3. Enter the contract address, `0x07C152A6ab577E8F78e3bedE502D79652787A9FC`, which will automatically display the **Token Symbol** and the decimal points, as shown in the following screenshot:



Import tokens

Custom token

Token detection is not available on this network yet. Please import token manually and make sure you trust it. Learn about [scams and security risks](#).

Token contract address

0x07C152A6ab577E8F78e3bedE502D79

Token symbol [Edit](#)

MET

Token decimal

Add custom token

Figure 18.5: Custom token in MetaMask

4. Press **Add custom token**, and then press **Import tokens** on the next screen. Now we have **100 MET** in our MetaMask wallet! Now let's send 10 MET to another account on the Sepolia network.
5. Click on **Send**, then enter the target account's details.
6. Enter **10 MET** and click **Next**.
7. Then click **Confirm**, and the transaction will be processed.
8. Now, coming back to the Remix IDE, we can see that our balance has been reduced by 10 MET. We can see this by passing the address of the token holder, `0xb838042b89ebdfb1d5ed0de32323f798eb5dc96e`, and clicking the **balanceof** button. Similarly, we can check the balance of the target account, `0xb838042b89ebdfb1d5ed0de32323f798eb5dc96e`, to which we transferred 10 MET, which is now **10 MET**. You can see this by passing `0xb838042b89ebdfb1d5ed0de32323f798eb5dc96e` to the **balanceof** function.

We can see all the transfers of our ERC-20 token, MET, on Etherscan: <https://sepolia.etherscan.io/token/0x07c152a6ab577e8f78e3bede502d79652787a9fc>.

With this, we have covered how to create an ERC-20 token from scratch and deploy it on the Ethereum blockchain.



Note that OpenZeppelin is an open source framework for building secure smart contracts and has many pre-built token standards that can be easily used by simply importing them in your Solidity code. More information is available here: <https://docs.openzeppelin.com/contracts/>

Let's now have a look at some of the novel concepts that are emerging due to the remarkable success of tokenization and the blockchain ecosystem in general.

Emerging concepts

With the advent of blockchain and tokenization, several new concepts have emerged over the last few years. We will introduce some of them now.

Tokenomics/token economics

Tokenomics or token economics is an emerging discipline that is concerned with the study of economic activity, economic models, and the impact of tokenization. It deals with the goods and assets that have been tokenized and the entities that are involved in the entire process of token issuance, sale, purchase, and investment.



L You might have heard another term, **cryptoeconomics**, which is a related but slightly different term. Cryptoeconomics is concerned with the same topics, but it is a superset of tokenomics. In other words, tokenomics is a subset of cryptoeconomics. Tokenomics is only concerned with tokens and tokenization ecosystems, but does not include the broader blockchain networks, protocols, and cryptocurrencies.

With the use of the **proof of work (PoW)** mechanism in Bitcoin, it was demonstrated for the first time that computer protocols can be designed in such a way that attacking a system does not result in achieving an uneven advantage or commercial benefit. This concept then further matured into what we call today cryptoeconomics. This can also be understood as a combination of economics, game theory, and cryptography.

Token engineering

Token engineering is an emerging concept that is looking at tokenization from an engineering perspective and is striving to apply the same rigor, systems thinking, and mathematical foundations to tokenization and blockchain in general that a usual engineering discipline has. This subject is still in its infancy; however, good progress has been made toward the development of this new discipline.



More information on token engineering can be found at <https://tokenengineeringcommunity.github.io/website/>

Token taxonomy

There is a lack of consistent taxonomy for tokens. As such, there are no clear standards defined on how to design and manage tokens. Also, it is not clear how to reuse an already existing and working token design, or if any exist at all.

Therefore, there is a need for a classification system that categorizes all different types of tokens according to the different attributes they have. There is also no universal classification of different attributes of tokens such as type, value, and economic attributes. Such a system would benefit the tokenization ecosystem tremendously.

Don't confuse this with the ERC standards we explained earlier; those are development standards, and they are certainly useful. But they are specific in scope and are not the universal classification of tokens.

Some work on this was started by Interwork Alliance by producing a **Token Taxonomy Framework (TTF)**. More details on this work can be found on their website: <https://interwork.org>.

Summary

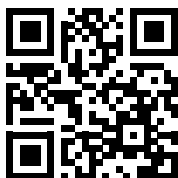
In this chapter, we covered tokenization and relevant concepts and standards. We also covered different types of tokens and related token standards. Moreover, a primer on trading and finance was also provided to familiarize readers with some standard finance concepts, which help us to understand the decentralized finance ecosystem too, as most terminology is borrowed from traditional finance.

Also, we introduced a practical example of how to create our own ERC-20-compliant token using the Ethereum platform. Finally, we introduced some emerging ideas related to tokenization.

In the next chapter, we will explore how blockchain can be used in business contexts, in addition to the cryptocurrency and contract deployment related contexts we have reviewed so far.

Join us on Discord!

To join the Discord community for this book – where you can share feedback, ask questions to the author, and learn about new releases – follow the QR code below:



<https://packt.link/ips2H>